Introduction to Linux

Cecina Babich Morrow

2024-03-01

Introduction

The following diagram displays the relationship between the hardware, kernel, and shell in a computer:

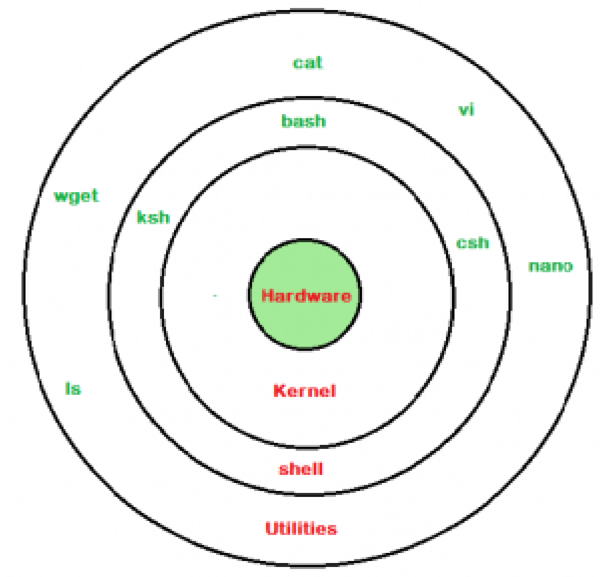

Figure 1: Diagram from Geeks for Geeks

We have the following components:

- Kernel: the foundation of the computer's operating system
  - Kernel options: Linux, MacOs, Windows
- Shell: the language you are speaking to the kernel – typically users interact with the shell, not the kernel directly
  - Shell options: Ksh, Zsh, Bash, cmd, Powershell
  - Bash (Bourne Again SHell) is the most common
- Programs to use the shell: what the user actually opens to interact with the shell (not actually shown on the diagram)
  - Options: Gnome Terminal, Konsole, MacOs Terminal, Git Bash, Powershell
- Utilities: programs that help the user interact with the shell – we will discuss many of these in detail throughout the rest of the portfolio

File system navigation

We must first be able to navigate ourselves around the folders and files on our computer. Here are a list of common commands for navigation:

- `pwd`: print the current working directory
- `cd`: change the current working directory
- `mkdir`: make a new directory
- `ls`: list the contents of a directory (see next section)

`ls`

`ls` is a highly useful command-line utility for listing the contents of a directory. Here are some common flags used with `ls`:

- `ls -l` gives you a long list of files in the directory, including the permissions, the number of links, the owner, the group, the size, the date, and the name of the file
- `ls -lt` gives you a long list of files in the directory, sorted by time
- `ls -ltr` gives you a long list of files in the directory, sorted by time in reverse order
- `ls -hl` gives you the file sizes in human-readable format

File permissions

When running `ls -l`, we can see the permissions of each file. For instance, if the permissions of a given file are `drwxr-xr-x`, this is how to interpret that:

- `d` means it is a directory (files start with `-` instead)
- The `d` or `-` is followed by 3 triples: what is the owner allowed to do, what is the group allowed to do, and what is everyone else allowed to do
  - `rwx` means that someone can read, write, and execute
  - Anything replaced with `-` is an action that that entity cannot do, e.g. `r-x` means that the entity can read and execute, but not write

To change the permissions of a file, you can use the `chmod` command. For example `chmod +x` will add execute permissions to a file for the owner, group, and everyone else.

Working with files

We will start with common Linux commands for working with files:

- `cp`: copies a file to a new destination (will overwrite if there is an existing file at that destination)
- `rm`: deletes a file
- `mv`: moves a file to a new destination (basically renaming)

Some exercises

Can you `mv` one file to more than one destination?

This is not possible. If you try to run `mv file1 file2 file3`, you will get the error `mv: target 'file3' is not a directory`.

What happens if you give cp no destination?

You get the following error:

```
cd ./intro-to-command-line/command-line-files/sandbox
cp data4
```

```
## cp: missing destination file operand after 'data4'
## Try 'cp --help' for more information.
```

Can you cp, but without being inside the folder it is located in?

Yes!

```
cd ./intro-to-command-line/command-line-files/sandbox
cp new_folder_copy/file2 folder_copy_file2
```

Can you move a folder?

Yes!

```
mv folder_copy new_folder_copy
```

How does moving a file onto a folder behave?

```
mv data2 new_folder_copy
cd new_folder_copy/
ls
```

The results are:

```
copy-of-data2-but-HERE  data2  file1  file2  file3
```

Can you mv the folder you are actually in?

You get the following error:

```
cd ./intro-to-command-line/command-line-files/sandbox/folder_with_files_inside
mv . ../new_folder_copy2
```

```
## mv: cannot move '.' to '../new_folder_copy2': Device or resource busy
```

Viewing the contents of files

Some options for viewing the contents of files:

- `cat`: prints the contents of a file to the terminal
- `head` & `tail`: prints the first or last 10 lines of a file, respectively (you can change the number of lines printed by running `head -n 5 file.txt`, for example)
- `tac`: prints the contents of a file in reverse by line (i.e. the last line is printed first)
- `rev`: prints the contents of a file with the contents of each line reversed by character
- `uniq`: prints the contents of a file with adjacent identical lines collapsed to one
- `sort`: prints the contents of a file sorted
- `less`: allows you to scroll through the contents of a file. If you want to find a specific string, you can type `/` and then the string you want to find, and then press `n` to find the next instance of that string

Editing files

Some popular text editors:

- `nano`: a simple text editor
- `micro`: a modern version of `nano`
- `emacs`: a powerful text editor with higher learning curve than `nano` / `micro`
- `jove`: a more minimal version of `emacs`
- `vim`: a powerful text editor with a steep learning curve that uses lots of keyboard shortcuts

Searching

Searching in files

`grep` is a command-line utility for searching plain-text data sets for lines that match a regular expression.

Regular expressions

Regular expressions are a way to describe patterns in text. The website regex101 has resources about regular expressions and the ability to test them.

Some useful regular expression syntax:

- `.` matches any character
- `^` matches the start of a line
- `$` matches the end of a line
- `*` matches the preceding element zero or more times
- `+` matches the preceding element one or more times
- `?` matches the preceding element zero or one time
- `[ab]` matches either "a" or "b"

`grep` flags

Here are some common flags used by `grep`:

- `grep -c`: prints the number of lines that match the pattern
  - If you have multiple targets, it will print the number of matches for each target, e.g. `grep -c "villain" king-lear.txt julius-caesar.txt` yields

    ```
    king-lear.txt:20
    julius-caesar.txt:4
    ```
- `grep -i`: makes the search case-insensitive
  - Combining `grep -ci` will give you the number of matches case-insensitive
- `grep -n`: prints the line number of each matching line
- `grep -cih`: prints the number of matches case-insensitive and suppresses the file names
- `grep -A2`: prints the line matching the pattern and the two lines following it
- `grep -B2`: prints the line matching the pattern and the two lines preceding it
- `grep -C2`: prints the line matching the pattern and the two lines preceding and following it (alternatively, you could run `grep -A2 -B2`)
- `grep -w`: matches only whole words
- `grep -l`: prints the names of files with matching lines
- `grep -r`: searches recursively through directories

You can also use the wildcard `*` to search within files that match a particular pattern, e.g. `grep sparrow a*` will search for the word "sparrow" in all files that start with "a".

Searching exercises

To determine which plays contain the word "squirrel", we can run:

```
cd ./intro-to-command-line/command-line-files/some_plays
grep -l "squirrel" *
```

```
## a-midsummer-nights-dream.txt
## romeo-and-juliet.txt
```

Similarly, we can list the plays containing "toasted cheese":

```
cd ./intro-to-command-line/command-line-files/some_plays
grep -l "toasted cheese" *
```

```
## henry-vi-part-2.txt
## king-lear.txt
## merry-wives-of-windsor.txt
```

Now we want to count the number of plays containing the word "confidence":

```
cd ./intro-to-command-line/command-line-files/some_plays
grep -l "confidence" * | wc -l
```

```
## 12
```

Note that we are using the `|` operator, which is described in more detail in the Pipes section below. We list the plays containing "confidence" using `grep -l` and pipe the results to `wc` to count the number of plays. The command `wc` gives the number of lines, words, and characters in a file, and the `-l` flag extracts only the number of lines, which is in this case the number of plays containing "confidence".

To get the line number of the word "folly" in *Hamlet*:

```
cd ./intro-to-command-line/command-line-files/some_plays
grep -n "folly" hamlet.txt
```

```
## 4761:    But that this folly douts it.
```

We want to search for the word "asleep" in plays that have the word "and" in their title:

```
cd ./intro-to-command-line/command-line-files/some_plays
grep -lc "asleep" *and*
```

```
## antony-and-cleopatra.txt
## romeo-and-juliet.txt
## titus-andronicus.txt
```

To find how many times Lady Macbeth speaks in *Hamlet*, we use the fact that when someone is speaking, their name is in all caps and it starts the line:

```
cd ./intro-to-command-line/command-line-files/some_plays
grep -c "^LADY MACBETH" macbeth.txt
```

```
## 60
```

Note that ˆ represents the start of a line in regular expressions. We find that she speaks 60 times in the play.

To find out how many times both Romeo and Juliet speak in *Romeo and Juliet*, we can run:

```
cd ./intro-to-command-line/command-line-files/some_plays
grep -c "^\(ROMEO\|JULIET\)" romeo-and-juliet.txt
```

```
## 283
```

This command uses the `\|` character to indicate "or" in the regular expression. We find that the two characters speak a combined number of `283` times.

Searching for files

`find` is a command-line utility for searching for files in a directory hierarchy. `find` has the following syntax: `find [where to look] [what type of file] ["the match you are looking for, in quotes"]`.

Redirection

Programs can write to the output stream and the error stream. The output stream is where the program writes its normal output, and the error stream is where the program writes its error messages.

The `>` operator redirects the output stream to a new file (or overwrite an existing file). The `>>` operator appends the output stream to a file instead.

Redirection exercises

Make a new file called "first_50_lines.txt" of the first 50 lines of *Romeo and Juliet*:

```
cd ./intro-to-command-line/command-line-files/some_plays
head -n 50 romeo-and-juliet.txt > first_50_lines.txt
```

Make another new file, containing the last 86 lines, of all the plays:

```
cd ./intro-to-command-line/command-line-files/some_plays
tail -n 86 * > last_86_lines.txt
```

Make a file containing all the lines, from all the plays, containing the word "Queen":

```
cd ./intro-to-command-line/command-line-files/some_plays
rm queen_lines.txt # remove the file if it already exists
grep -h "Queen" * > queen_lines.txt
```

Note the use of the `-h` flag to suppress the file names in the output.

Create a file containing the total number of lines spoken by Theseus, Oberon, and Lysander in *A Midsummer Night's Dream*:

```
cd ./intro-to-command-line/command-line-files/some_plays
grep -c "^\(THESEUS\|OBERON\|LYSANDER\)" a-midsummer-nights-dream.txt > midsummer_line_count.txt
```

Add the final 100 lines of *The Tempest* to the `midsummer_line_count.txt` file:

```
cd ./intro-to-command-line/command-line-files/some_plays
tail -n 100 the-tempest.txt >> midsummer_line_count.txt
```

Pipes

The pipe operator `|` sends the output of one program to the input of another program.

Pipe exercises

How many .txt files are in the some_plays folder?

```
cd ./intro-to-command-line/command-line-files/some_plays
ls | grep -c ".txt"
```

```
## 31
```

Which play has exactly 5730 lines?

```
cd ./intro-to-command-line/command-line-files/some_plays
wc -l * | grep "5730"
```

```
##    5730 othello.txt
```

How many lines, in all the plays, have both the words "true" and "love"?

```
cd ./intro-to-command-line/command-line-files/some_plays
cat * | grep -c "true.*love\|love.*true"
```

```
## 35
```

This code concatenates all of the plays together and then counts lines that have both words in them. Note that the regular expression `true.*love\|love.*true` matches lines that have both words in them, in either order.

What is the total line count of every file in the folder?

```
cd ./intro-to-command-line/command-line-files/some_plays
cat * | wc -l
```

```
## 121919
```

What is the 629th line of *Pericles*?

```
cd ./intro-to-command-line/command-line-files/some_plays
cat pericles.txt | head -n 629 | tail -n 1
```

```
##  Whose towers bore heads so high they kiss'd the clouds,
```

What is the 97th line of *Othello* backwards?

```
cd ./intro-to-command-line/command-line-files/some_plays
cat othello.txt | head -n 97 | tail -n 1 | rev
```

```
## kram llahs uoY .dewollof ylurt eb tonnaC
```

What is the 107th from last line of *Anthony and Cleopatra*?

```
cd ./intro-to-command-line/command-line-files/some_plays
cat antony-and-cleopatra.txt | tail -n 107 | head -n 1
```

```
## [She closes Cleopatra's eyes.]
```

What are the five longest plays in order of line count?

We need to make sure we aren't including the line with the total resulting from `sort`:

```
cd ./intro-to-command-line/command-line-files/some_plays
wc -l * | sort -n | tail -n 6 | head -n 5
```

```
##    5563 king-lear.txt
##    5619 coriolanus.txt
##    5730 othello.txt
##    5870 antony-and-cleopatra.txt
##    5877 hamlet.txt
```

Combining pipes and redirection

We can combine the use pipes and redirection to create a files from the result of piping one command to another. For example, the following code creates a new file containing every line in *The Count of Monte Cristo* that contains the word "hill":

```
cd ./intro-to-command-line/command-line-files/alexandre-dumas
grep -w "hills" The-Count-Of-Monte-Cristo.txt > hill-file.txt
```

Shell scripts

Rather than typing commands into the terminal interactively, we can save our bash commands into a script file for execution. To run the script, we can use the command `bash [script_name].sh`. By convention, the first line of the script should be `#!/bin/bash`, which tells the shell that this is a bash script and should be interpreted as such.

For more information about bash scripting, see the Ryans Tutorials Bash Scripting Tutorial.